DC Motor with Shaft Encoder

Learning Objectives

By the end of this laboratory experiment, the experimenter should be able to:

- Explain how an encoder operates and how it can be use determine rotational speed and angle of a motor shaft
- Explain the concept of Pulse-Width Modulation to control the speed of a DC motor
- Interface the ATmega16 microcontroller to an encoder

Components

Qty.	Item
1	ATmega16 microcontroller, STK500, serial port cable, two 2-pin jumpers, one 10-pin jumper
1	DC motor mounted in laser-cut acrylic stand with encoder wheel and two optical encoder pickup boards
1	IRL520 power MOSFET
1	1N4148 diode
1	1M Ω resistors
1	Solderless breadboard

Introduction

In this lab you will investigate how rotational speed and rotational angle can be determined using a rotary encoder. A rotary encoder consists of a disk with alternating opaque and clear radial regions. In operation, a light source is positioned on one side of the disk, and a photosensitive device, such as a phototransistor is positioned on the other side of the disk. As the disk rotates, the passage of the opaque and clear regions of the encoder disk alternately block and allow light to impinge on the receiver, which produces corresponding voltage pulses. The rotational speed of the encoder disk can be determined by counting pulses during a known time period. The angle of rotation corresponds directly to the number of pulses, since the number of pulses per revolution is constant.

Circuit Board – Optical Encoder Board

As mentioned above, an optical encoder works by pairing some type of light emitting device with a complementary light sensing device. Similarly one could use a magnet and a magnetic field sensing element by mounting one of the two on the rotating member and the other on a stationary point within the range of the sensing element.

Figure 1 below shows the circuit diagram for the optical encoder system you will use in this experiment. Here, an infrared (IR) LED emits light towards a photosensitive transistor (phototransistor) mounted in one package, separated by a gap. Every time that the phototransistor detects a on-off-on or off-on-off cycle, receiver logic notes/counts the event. By knowing the physical characteristics of the interrupting element (such as number of windows in an opaque wheel per revolution), one can compute the rotational speed of the device on which it is mounted.

The output of the phototransistor does not change sharply from a lower to higher voltage (or vice-versa) as the incoming light beam is interrupted because of diffraction, nor does it necessarily output a voltage that can immediately or readily identified as a logic-high or logic-low signal. For this reason, the output of the transistor is fed into a device called a *comparator* that generates only logic-level

outputs given an analog input. Voltage inputs above a settable value generate a logic-high output, and voltages below that value result in a logic-low output. This is one of the most common uses of a comparator, to "square-up" an analog input signal, producing a square wave on its output. Figure 2 below shows the signal before and after the comparator.

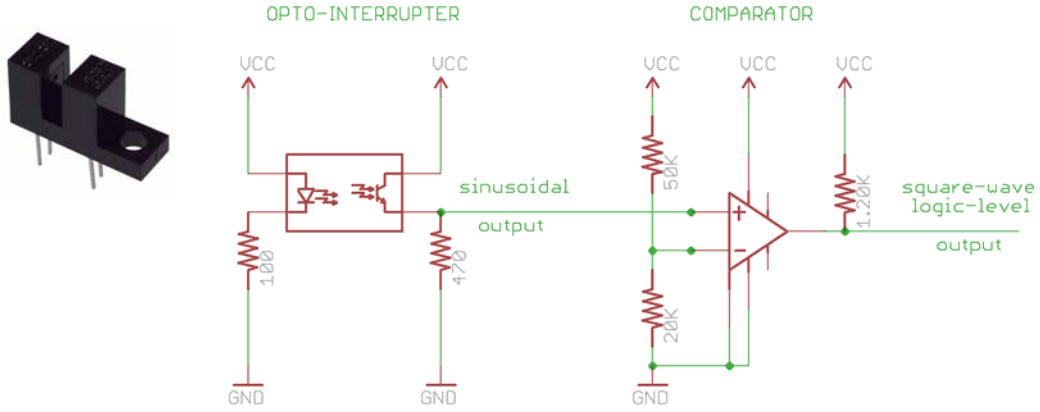

Figure 1. Circuit diagram for opto-interrupter based encoder board. (image of slotted opto-interrupter switch from http://media.digikey.com/photos/Omron%20Elect%20Photos/EE-SK3W-B.jpg)

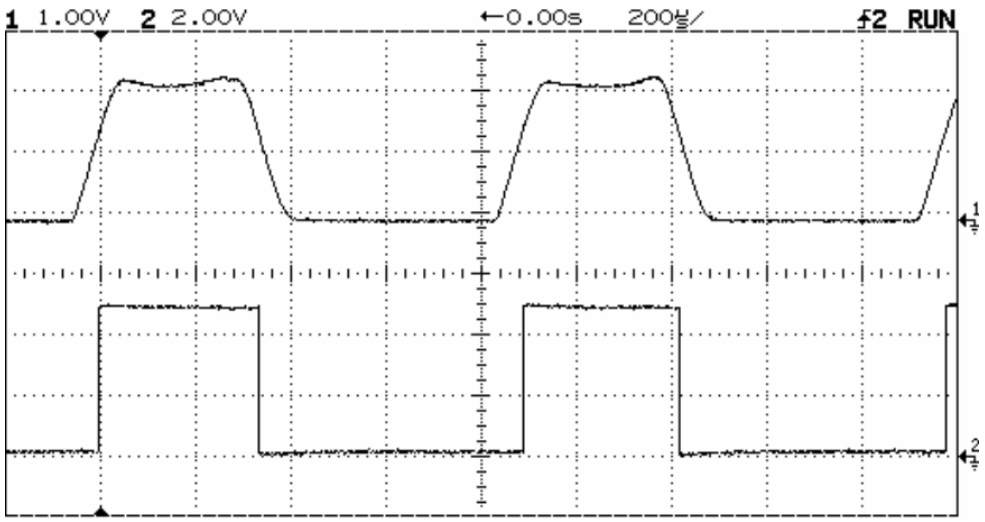

Figure 2 A input sine wave being squared-up into a square wave output

The comparison value is set in the above circuit with a simple voltage divider on the negative (-) terminal of the comparator. ***What is this comparison value given the resistor values above, assuming VCC=5V ?***

The board containing the opto-interrupter and comparator circuit has a three-wire interface, two of which supply power and ground to the circuitry on the board (the red and black wires, respectively), and a white wire that returns the output square-wave, which represents the light-level through the encoder window.

Procedure

Function Generator Output to Control Duty Cycle

1. Set up the function generator (FG) to output a 1 kHz square wave (remember to set the output termination to HIGH Z). Look at the signal on the 'scope.
2. Set the amplitude to 8 V p-p
3. Offset the waveform by 4 V, so that you have a 0 to 8 V square wave.
4. Select the '% Duty' function on the FG by pressing the Shift key, then the Offset key.
5. Rotate the knob and see what happens to the output waveform when you vary the duty cycle. **What are the limits you can set the duty cycle to? What does "duty cycle" mean? Describe in your own words.**

DC Motor Speed Control Using the Function Generator to Control the Duty Cycle

6. Build the circuit shown in Figure 3. Don't forget the diode. **Important note**: MOSFET's are very sensitive to static electricity. Make sure that you are not carrying static charge before you handle these devices. It is best to work on a properly grounded anti-static surface with an anti-static bracelet on your wrist. If these precautions are not available, then discharge yourself by touching a grounded metal surface (such as the frame of the bench) before you handle a MOSFET. Always handle a MOSFET by its large metal tab and NOT by its leads.

   The MOSFET you will be using in this lab is the IRL520. It is specifically designed to be fully turned on by *logic-level* circuits (5 V), which makes it ideal for controlling medium power devices, such as dc motors, using a microcontroller. Its package style is an industry standard TO-220. This package is somewhat awkward to use in a solderless breadboard, because its leads are so large. To avoid damaging the solderless breadboard, insert the IRL520 so that its metal tab is ***parallel*** to one of the 5-hole rows, but with each lead in a separate row. To do this, you will have to bend the leads slightly (see Figure 4). The 1 MΩ resistor is used to make sure that charge can bleed out of the gate to ground to turn off the MOSFET in the event that the microcontroller pin it connects to inadvertently changes from being an output (with logic high asserted) to an input (high impedance).

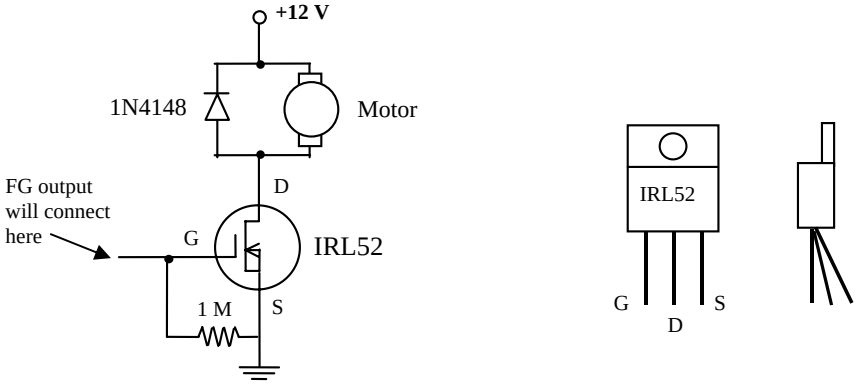

Figure 3. DC Motor Driver Circuit with IRL520 power MOSFET. MOSFET's are very static sensitive, so handle them by the metal tab.

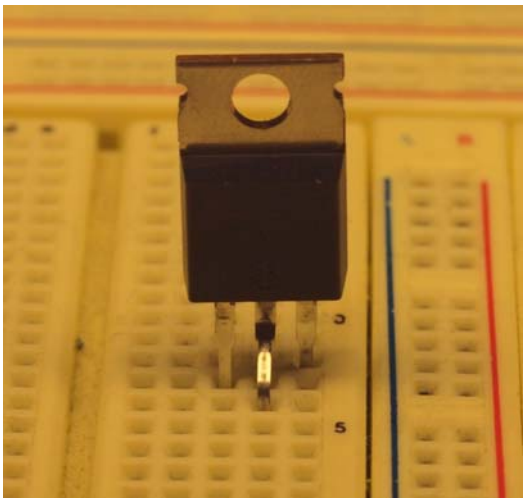

Figure 4. How the IRL520 MOSFET should be plugged into the breadboard

Why is the diode needed across the motor leads? If you are unsure, ask your laboratory assistant to explain why the diode is important when interfacing motors. (Note: use of the 1N4148 diode is a concession to the limitation of the solderless breadboard to accommodate large wire diameter. It would be more appropriate to use a standard rectifier diode with larger forward current capacity, such as a 1N4003, however, the lead diameter for the 1N4003 is too large for the solderless breadboard, which is suited for 22 ga. solid core wire. Leads with significantly larger diameters when forced into the breadboard holes will permanently bend the internal contacts. Subsequent use of the breadboard hole with 22 ga. wire can then lead to unreliable contacts. A work-around to this problem is to solder 22 ga. wire to the oversize lead. The current draw of the motors used in this lab is low enough that the 1N4148 will marginally suffice Figure 5 shows the way to plug the MOSFET in without damaging the breadboard. Figure 7 provides more information on rectifier diodes.

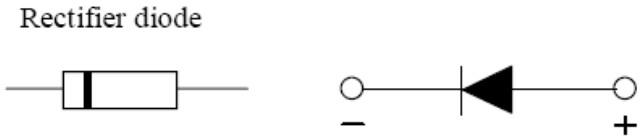

Figure 5. Rectifier Diode Representations. The band on the diagram to the left corresponds to the vertical line on the schematic symbol for the diode on the right and can be used to determine the polarity of the diode. A diode acts like a check-valve for current flow. The diode is said to be 'forward-biased' when its anode voltage is higher than its cathode voltage. (Which lead is the anode, and which is the cathode?) The 'check-valve' opens up when the forward-bias voltage is approximately 0.6 V – 1 V.

7. With the FG set up from Step 5, clip the red mini-hook lead from the function generator cable to the gate of the IRL520 and the black mini-hook lead to the common ground. Vary the duty cycle of the signal from the FG, and observe what happens to the motor. **Explain why the motor speed varies with the duty cycle.**

Connecting the Motor Encoders to the STK500 Development Board

8. Unhook the red lead of the FG from the gate of the IRL520.
9. There are three wires coming out from each of the two opto-encoder boards. The red wire and black wire should be connected to +5V and Ground from the STK500 interface board respectively. Figure

6 below shows how the wires should be connected between STK500 and the opto-encoder boards. This will power the encoder's optointerrupter and provide 0-5V signals to the left and right encoder output signal as the encoder wheel turns. Look at both of the outputs on the oscilloscope. Since the signals from channels A and B are out of phase, **explain how the two signals from channel A and channel B can be used to determine the direction that the motor spins.**

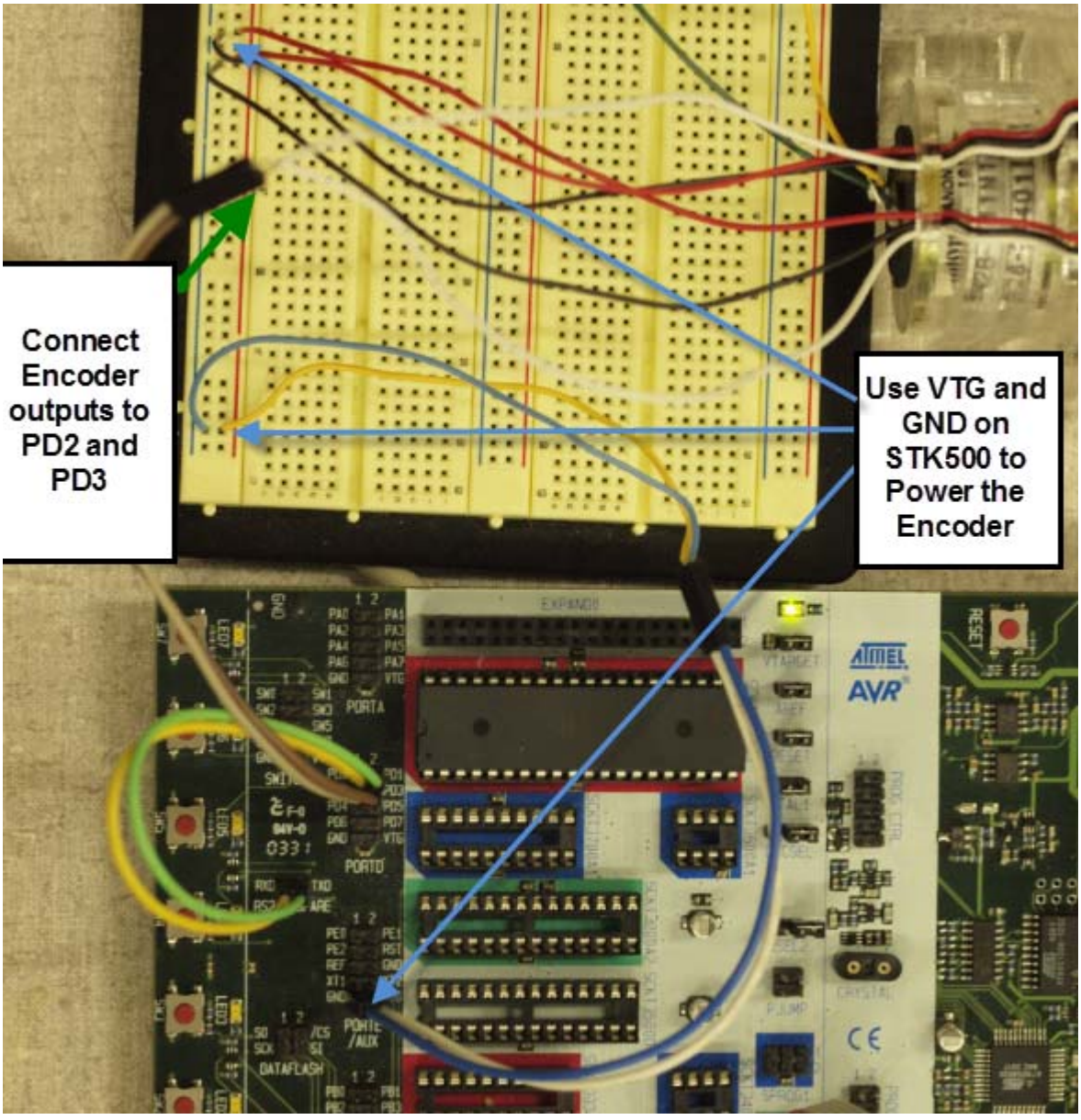

Figure 6. Encoder to STK500 connection. DO NOT make the PD2, PD3 connections until step 13.

10. Now, connect the 'scope to the Ch. A and Ground pins of the encoder.
11. Power the STK500.
12. Re-connect the red mini-hook of the FG to the gate of the IRL520, and run the motor as in Step 7. Observe the output of channel A on the 'scope. **Determine the speed of the motor at five different duty cycles that span the full range that the FG can output. Plot motor speed vs. % duty cycle in your report, and discuss your results.**

ATmega16 Measurement of Encoder Pulses

13. Disconnect the red mini-hook of the FG and turn off the 12 V power supply. Make sure that the ATmega16 controller is **OFF**. Connect the output of the left encoder to pin PD2 of the ATmega16, and right encoder to pin PD3 (The left encoder output can really go to any other unused input pin, however you need to keep track of which pin, and make necessary changes in your code).
14. At this time, connect the serial cable to the COM port on your computer and to the CTRL RS232 COM port connector on the STK500.
15. After **DOUBLE CHECKING** all of the connections to the ATmega16, turn on the 12V power supply to power up the device. Check to make sure that the power LED on the STK500 is on (it should turn red, then yellow, then flash green, and finally stay green). If the light does not come on, see the TA for help. Do not proceed if the board does not power up!
16. We are going to use an interrupt service routine (ISR) to keep track of the encoder pulses from the motor. An ISR is a special kind of function often used in applications with microcontrollers, which is executed when a specific kind of event occurs. Such events might be a rising edge (low to high transition), a falling edge (high to low transition) on a specific pin, an overflow of a counter, etc. When one of these events occurs, the regular program operation is "interrupted", and the program jumps to the ISR code to "handle" the situation triggered by the event. After the ISR instructions are completed, the program returns to what it was previously doing. Interrupts are powerful tools for embedded systems programming. They allow the microcontroller to perform other tasks (such as send or receive data across the serial port) without having to be tied up in waiting for an input state to change. ISRs must be kept short, and care must be taken in their use, so that timing and reliability are not compromised. For more information on interrupts, see:

    http://www.nongnu.org/avr-libc/user-manual/group__avr__interrupts.html

    In this lab we will have the encoder Phase A trigger an external interrupt, while Phase B is routed to another input pin. The ISR will check the state of the Phase B input pin to determine the direction of the encoder and then increment or decrement the encoder's position variable. With the encoder position known, we can count the number of encoder ticks within a given amount of time. This can be translated into the motor's speed in encoder ticks per second. Finally, we will display the values for encoder position and speed by sending them to the serial port. You can use a terminal program such as HyperTerminal to view the output from the serial port. If you use HyperTerminal, configure the COM port to 9600,8,N,1,N, and the ASCII settings should ***not*** append line feeds to incoming line ends.
17. Download and use the source files from the ME 106 website. Open a COM control or HyperTerminal window to display output from your program.
18. Reconnect the red mini-hook of the FG to the gate of the IRL520. Explain how the program works. **Compare the speed you measure with the 'scope with the value your program outputs. How well do these values agree?**
19. Suppose the encoder were mounted to the shaft of a motor or the wheel of a vehicle. If the diameter of the wheel is 3 inches, write a program that will indicate the rotational speed of the wheel and tell how far the vehicle has traveled. How accurately can you calculate the speed and distance? Quantify your answer. *HINT: The encoder has 18 slots per revolution.*

Some Notes about Reading Encoders and Measuring Speed:

When measuring speed using encoders, you can go about it several ways: you can either fix the time period and count the number of pulses that occur during that period, or fix the number of pulses and measure the amount of time between them. In the code for this lab, we take the first approach by counting encoder pulses (which corresponds to the amount of shaft rotation) in a set amount of time. To do this, we utilize a hardware timer to give us a fixed time interval, and we use an interrupt to count the number of encoder ticks that take place during the interval. It is possible to do the counting with a hardware counter, and thus save the ATmega16 the trouble of breaking out of the main loop to handle incrementing the count. Such an approach is fine if all we are measuring is speed, but by doing the counting in software, we can add logic that detects which *direction* the encoder is turning. More powerful microcontrollers often have full quadrature encoder counters built in, which unburdens the processor and programmer from having to keep track of the count and determine direction.